IT314: Software Engineering

Lab: 8

ID: 202201216

Name: Jeet Patel

Question: 1

1. Program Specification

Input: Triple of day, month, and year

Input ranges:
1 <= month <= 12
1 <= day <= 31
1900 <= year <= 2015

Output: Previous date or "Invalid date"

2. Test Suite

2.1 Equivalence Partitioning

Valid Partitions:

- Normal days (not month end or year end)
- Month end (not year end)
- Year end (December 31)
- Leap year February 29

Invalid Partitions:

- Invalid month (< 1 or > 12)
- Invalid day (< 1 or > max days in month)
- Invalid year (< 1900 or > 2015)
- Invalid day for specific month (e.g., February 30)

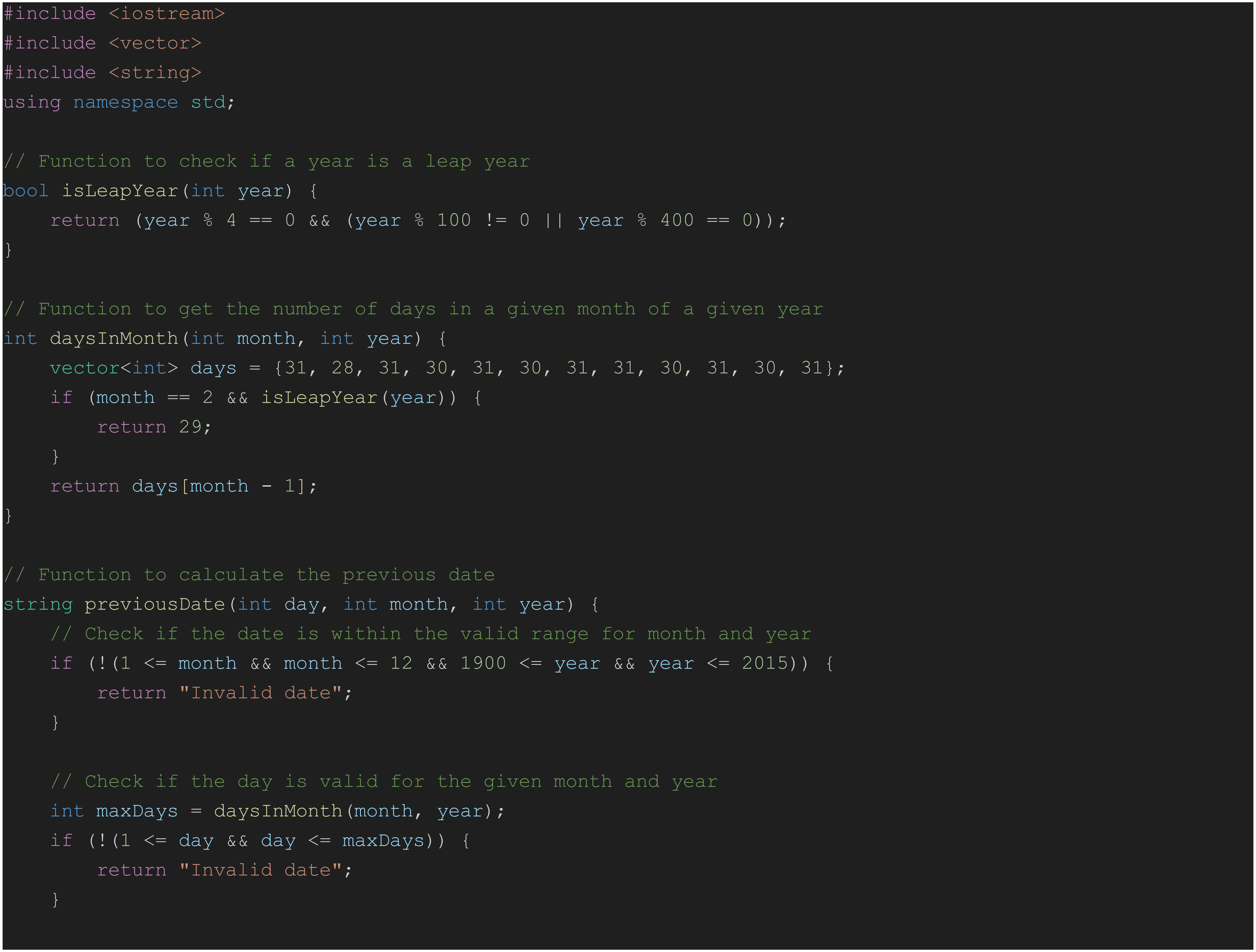
2.2 Boundary Value Analysis

- First day of year: January 1, YYYY
- Last day of year: December 31, YYYY
- First day of month: DD 1, MM
- Last day of month: DD 30/31, MM (28/29 for February)
- Minimum valid year: 1900
- Maximum valid year: 2015

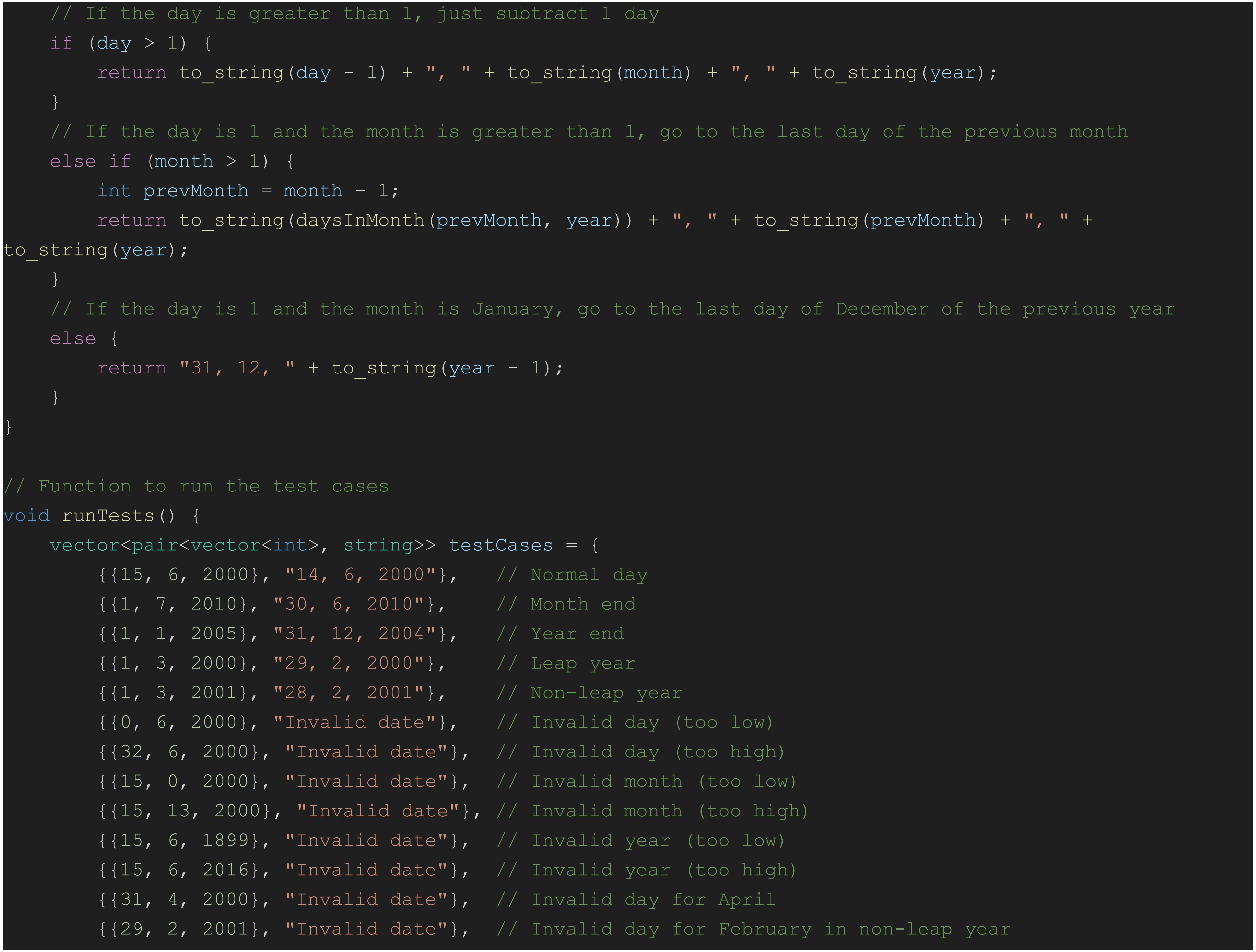
2.3 Test Cases

Input Data	**Expected Output**	**Remarks**
a, b, c	An Error message	Invalid input format
15, 6, 2000	14, 6, 2000	Normal day
1, 7, 2010	30, 6, 2010	Month end
1, 1, 2005	31, 12, 2004	Year end
1, 3, 2000	29, 2, 2000	Leap year
1, 3, 2001	28, 2, 2001	Non-leap year
0, 6, 2000	Invalid date	Invalid day (too low)
32, 6, 2000	Invalid date	Invalid day (too high)
15, 0, 2000	Invalid date	Invalid month (too low)
15, 13, 2000	Invalid date	Invalid month (too high)
15, 6, 1899	Invalid date	Invalid year (too low)
15, 6, 2016	Invalid date	Invalid year (too high)
31, 4, 2000	Invalid date	Invalid day for April
29, 2, 2001	Invalid date	Invalid day for February in non-leap year
1, 1, 1900	31, 12, 1899	Boundary: Minimum valid year - 1
31, 12, 2015	30, 12, 2015	Boundary: Maximum valid year
1, 1, 2000	31, 12, 1999	Boundary: First day of year
31, 12, 2000	30, 12, 2000	Boundary: Last day of year
1, 5, 2000	30, 4, 2000	Boundary: First day of month
31, 5, 2000	30, 5, 2000	Boundary: Last day of 31-day month
30, 4, 2000	29, 4, 2000	Boundary: Last day of 30-day month
29, 2, 2000	28, 2, 2000	Boundary: Last day of February in leap year
28, 2, 2001	27, 2, 2001	Boundary: Last day of February in non-leap year
1, 3, 2000	29, 2, 2000	Boundary: First day of March in leap year
1, 3, 2001	28, 2, 2001	Boundary: First day of March in non-leap year

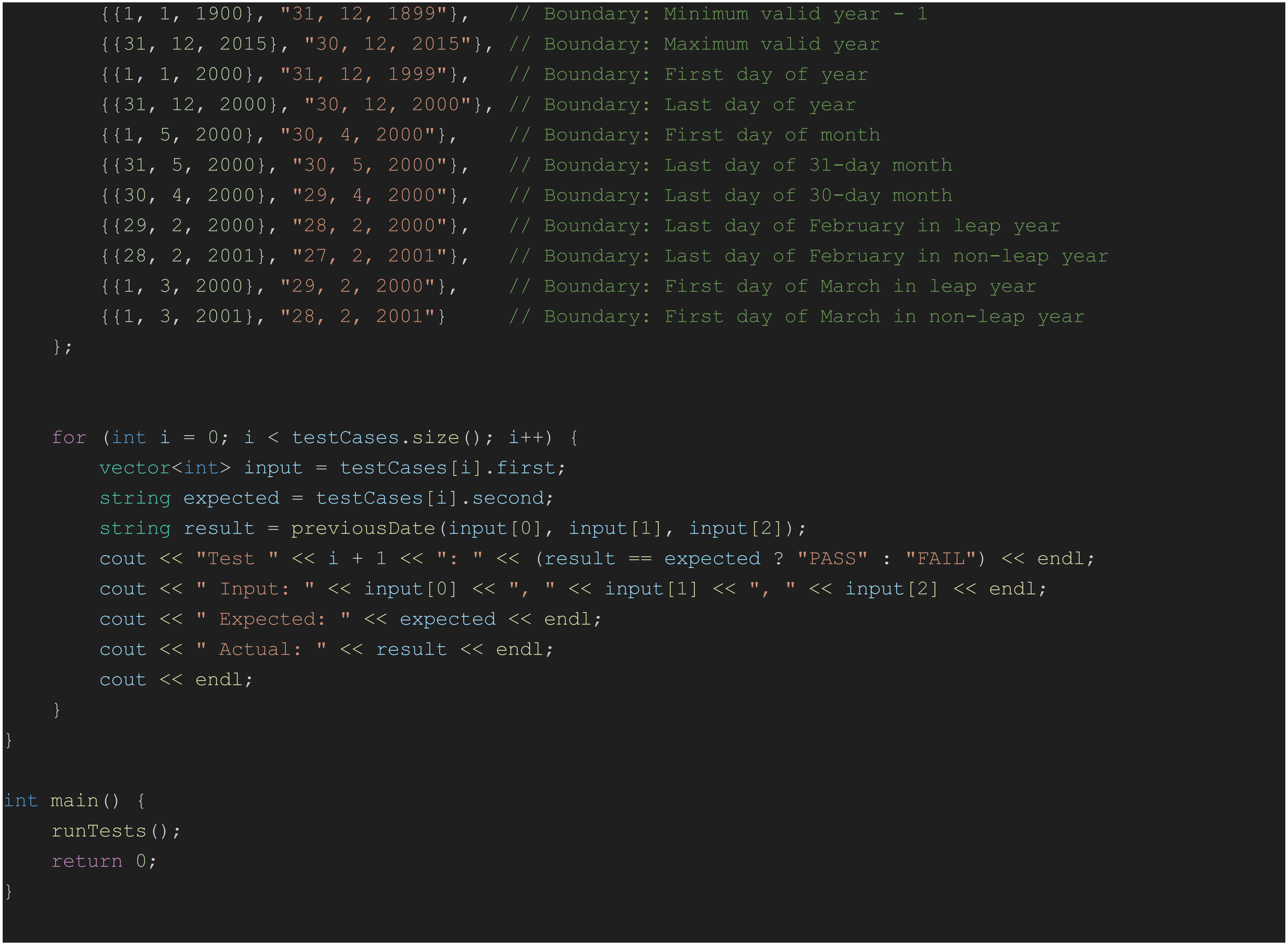
3. C++ Code (Functionality & Testing):



```cpp
#include <iostream>
#include <vector>
#include <string>
using namespace std;

// Function to check if a year is a leap year
bool isLeapYear(int year) {
    return (year % 4 == 0 && (year % 100 != 0 || year % 400 == 0));
}

// Function to get the number of days in a given month of a given year
int daysInMonth(int month, int year) {
    vector<int> days = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    if (month == 2 && isLeapYear(year)) {
        return 29;
    }
    return days[month - 1];
}

// Function to calculate the previous date
string previousDate(int day, int month, int year) {
    // Check if the date is within the valid range for month and year
    if (!(1 <= month && month <= 12 && 1900 <= year && year <= 2015)) {
        return "Invalid date";
    }

    // Check if the day is valid for the given month and year
    int maxDays = daysInMonth(month, year);
    if (!(1 <= day && day <= maxDays)) {
        return "Invalid date";
    }
```



```
    // If the day is greater than 1, just subtract 1 day
    if (day > 1) {
        return to_string(day - 1) + ", " + to_string(month) + ", " + to_string(year);
    }
    // If the day is 1 and the month is greater than 1, go to the last day of the previous month
    else if (month > 1) {
        int prevMonth = month - 1;
        return to_string(daysInMonth(prevMonth, year)) + ", " + to_string(prevMonth) + ", " +
to_string(year);
    }
    // If the day is 1 and the month is January, go to the last day of December of the previous year
    else {
        return "31, 12, " + to_string(year - 1);
    }
}

// Function to run the test cases
void runTests() {
    vector<pair<vector<int>, string>> testCases = {
        {{15, 6, 2000}, "14, 6, 2000"},   // Normal day
        {{1, 7, 2010}, "30, 6, 2010"},    // Month end
        {{1, 1, 2005}, "31, 12, 2004"},   // Year end
        {{1, 3, 2000}, "29, 2, 2000"},    // Leap year
        {{1, 3, 2001}, "28, 2, 2001"},    // Non-leap year
        {{0, 6, 2000}, "Invalid date"},   // Invalid day (too low)
        {{32, 6, 2000}, "Invalid date"},  // Invalid day (too high)
        {{15, 0, 2000}, "Invalid date"},  // Invalid month (too low)
        {{15, 13, 2000}, "Invalid date"}, // Invalid month (too high)
        {{15, 6, 1899}, "Invalid date"},  // Invalid year (too low)
        {{15, 6, 2016}, "Invalid date"},  // Invalid year (too high)
        {{31, 4, 2000}, "Invalid date"},  // Invalid day for April
        {{29, 2, 2001}, "Invalid date"},  // Invalid day for February in non-leap year
```



```
        {{1, 1, 1900}, "31, 12, 1899"},   // Boundary: Minimum valid year - 1
        {{31, 12, 2015}, "30, 12, 2015"}, // Boundary: Maximum valid year
        {{1, 1, 2000}, "31, 12, 1999"},   // Boundary: First day of year
        {{31, 12, 2000}, "30, 12, 2000"}, // Boundary: Last day of year
        {{1, 5, 2000}, "30, 4, 2000"},    // Boundary: First day of month
        {{31, 5, 2000}, "30, 5, 2000"},   // Boundary: Last day of 31-day month
        {{30, 4, 2000}, "29, 4, 2000"},   // Boundary: Last day of 30-day month
        {{29, 2, 2000}, "28, 2, 2000"},   // Boundary: Last day of February in leap year
        {{28, 2, 2001}, "27, 2, 2001"},   // Boundary: Last day of February in non-leap year
        {{1, 3, 2000}, "29, 2, 2000"},    // Boundary: First day of March in leap year
        {{1, 3, 2001}, "28, 2, 2001"}     // Boundary: First day of March in non-leap year
    };


    for (int i = 0; i < testCases.size(); i++) {
        vector<int> input = testCases[i].first;
        string expected = testCases[i].second;
        string result = previousDate(input[0], input[1], input[2]);
        cout << "Test " << i + 1 << ": " << (result == expected ? "PASS" : "FAIL") << endl;
        cout << " Input: " << input[0] << ", " << input[1] << ", " << input[2] << endl;
        cout << " Expected: " << expected << endl;
        cout << " Actual: " << result << endl;
        cout << endl;
    }
}

int main() {
    runTests();
    return 0;
}
```

Question: 2

Problem: 1

```
int linearSearch(int v, int a[])
{
    int i = 0;
    while (i < a.length)
    {
        if (a[i] == v)
            return (i);
        i++;
    }
    return (-1);
}
```

Equivalence Partitioning	
Input Data	**Expected Outcome**
5, {1, 2, 3}	-1
2, {1, 2, 3}	1
-1, {-1, 0, 1}	0
1, {}	-1
4, {4}	0
1, {1, 2, 3}	0
3, {1, 2, 3}	2
null, {1, 2, 3}	An Error message
{1, 2, 3}, null	An Error message
Boundary Value Analysis	
Input Data	**Expected Outcome**
5, {}	-1
-2147483648, {-2147483648, 0, 2147483647}	0
2147483647, {-2147483648, 0, 2147483647}	2
1, {1, 2}	0
2, {1, 2}	1
4, {1, 2, 3}	-1
5, null	An Error message
{1, 2, 3}, {}	An Error message

Problem: 2

```
int countItem(int v, int a[])
{
    int count = 0;
    for (int i = 0; i < a.length; i++)
    {
        if (a[i] == v)
            count++;
    }
    return (count);
}
```

Equivalence Partitioning	
Input Data	**Expected Outcome**
5, {1, 2, 3}	0
2, {1, 2, 3}	1
-1, {-1, 0, 1}	1
1, {}	0
4, {4, 4, 4}	3
1, {1, 2, 3, 1, 1}	3
3, {1, 2, 3, 3, 3, 3}	4
null, {1, 2, 3}	An Error message
{1, 2, 3}, null	An Error message
Boundary Value Analysis	
Input Data	**Expected Outcome**
5, {}	0
-2147483648, {-2147483648, 0, 2147483647}	1
2147483647, {-2147483648, 0, 2147483647}	1
1, {1, 2}	1
2, {1, 2, 2}	2
4, {1, 2, 3}	0
5, null	An Error message
{1, 2, 3}, {}	An Error message

Problem: 3

```
int binarySearch(int v, int a[])
{
    int lo, mid, hi;
    lo = 0;
    hi = a.length - 1;
    while (lo <= hi)
    {
        mid = (lo + hi) / 2;
        if (v == a[mid])
            return (mid);
        else if (v < a[mid])
            hi = mid - 1;
        else
            lo = mid + 1;
    }
    return (-1);
}
```

Equivalence Partitioning	
Input Data	**Expected Outcome**
5, {1, 2, 3}	-1
2, {1, 2, 3}	1
1, {1, 2, 3}	0
3, {1, 2, 3}	2
4, {1, 4, 6, 8}	1
0, {0, 1, 2, 3}	0
100, {10, 20, 30, 100}	3
null, {1, 2, 3}	An Error message
{1, 2, 3}, null	An Error message
Boundary Value Analysis	
Input Data	**Expected Outcome**
5, {}	-1
-2147483648, {-2147483648, 0, 2147483647}	0
2147483647, {-2147483648, 0, 2147483647}	2
1, {1, 2}	0
2, {1, 2}	1
4, {1, 2, 3}	-1
5, null	An Error message
{1, 2, 3}, {}	An Error message

Problem: 4

```
final int EQUILATERAL = 0;
final int ISOSCELES = 1;
final int SCALENE = 2;
final int INVALID = 3;
int triangle(int a, int b, int c)
{
    if (a >= b + c || b >= a + c || c >= a + b)
        return (INVALID);
    if (a == b && b == c)
        return (EQUILATERAL);
    if (a == b || a == c || b == c)
        return (ISOSCELES);
    return (SCALENE);
}
```

Equivalence Partitioning

Input Data	Expected Outcome
3, 3, 2003	EQUILATERAL (0)
3, 3, 2002	ISOSCELES (1)
3, 4, 2005	SCALENE (2)
1, 2, 2003	INVALID (3)
1, 1, 2002	INVALID (3)
5, 1, 2001	INVALID (3)
2, 2, 2003	ISOSCELES (1)
2000, 1, 1	An Error message
1, 0, 1	An Error message

Boundary Value Analysis

Input Data	Expected Outcome
1, 1, 2001	EQUILATERAL (0)
1, 1, 2002	INVALID (3)
2, 2, 2004	INVALID (3)
2, 3, 2005	INVALID (3)
3, 4, 2007	INVALID (3)
1, 2, 2002	ISOSCELES (1)
1, 2, 2003	INVALID (3)
2000, 1, 1	An Error message
1, 1, 2000	An Error message

Problem: 5

```java
public static boolean prefix(String s1, String s2)
{
    if (s1.length() > s2.length())
        return false;
    for (int i = 0; i < s1.length(); i++)
    {
        if (s1.charAt(i) != s2.charAt(i))
            return false;
    }
    return true;
}
```

Equivalence Partitioning

Input Data	Expected Outcome
"pre", "prefix"	TRUE
"pre", "postfix"	FALSE
"prefix", "pre"	FALSE
"test", "test"	TRUE
"", "anything"	TRUE
"anything", ""	FALSE
"pre", "preparation"	TRUE
null, "prefix"	An Error message
"prefix", null	An Error message

Boundary Value Analysis

Input Data	Expected Outcome
"test", ""	FALSE
"a", "a"	TRUE
"a", "b"	FALSE
"", ""	TRUE
"start", "startmiddle"	TRUE
"longprefix", "short"	FALSE
"short", "longprefix"	TRUE
null, "anything"	An Error message
"anything", null	An Error message

Problem: 6

a) Identify the Equivalence Classes

1. **Equilateral Triangle**: All three sides are equal.
2. **Isosceles Triangle**: Exactly two sides are equal.
3. **Scalene Triangle**: No sides are equal.
4. **Right-Angled Triangle**: Satisfies a^2+b^2 = c^2.
5. **Invalid Triangle**: Does not satisfy the triangle inequality a+b > c.
6. **Non-positive Input**: One or more sides are non-positive.

b) Identify Test Cases to Cover the Equivalence Classes

Input Data	Expected Outcome	Equivalence Class
3.0, 3.0, 3.0	Equilateral	Equilateral Triangle
3.0, 3.0, 2.0	Isosceles	Isosceles Triangle
3.0, 4.0, 5.0	Scalene	Scalene Triangle
3.0, 4.0, 0.0	Invalid	Invalid Triangle
0.0, 0.0, 0.0	Invalid	Non-positive Input
5.0, 1.0, 1.0	Invalid	Invalid Triangle
3.0, 4.0, 6.0	Scalene	Scalene Triangle

c) Boundary Condition A + B > C (Scalene Triangle)

Input Data	Expected Outcome
2.0, 2.0, 3.99	Scalene
2.0, 2.0, 4.0	Invalid
2.0, 2.0, 4.01	Invalid

d) Boundary Condition A = C (Isosceles Triangle)

Input Data	Expected Outcome
3.0, 4.0, 3.0	Isosceles
3.0, 3.0, 3.0	Equilateral
3.0, 3.0, 4.0	Isosceles

e) Boundary Condition A = B = C (Equilateral Triangle)

Input Data	Expected Outcome
3.0, 3.0, 3.0	Equilateral
1.0, 1.0, 1.0	Equilateral
2.5, 2.5, 2.5	Equilateral

f) Boundary Condition A^2+B^2 = C^2 (Right-Angled Triangle)

Input Data	Expected Outcome
3.0, 4.0, 5.0	Right Angled
6.0, 8.0, 10.0	Right Angled
5.0, 12.0, 13.0	Right Angled

g) Non-Triangle Case

Input Data	Expected Outcome
1.0, 2.0, 3.0	Invalid
1.0, 2.0, 4.0	Invalid
1.0, 1.0, 2.0	Invalid

h) Non-Positive Input

Input Data	Expected Outcome
0.0, 1.0, 1.0	Invalid
-1.0, 1.0, 1.0	Invalid
1.0, 0.0, 1.0	Invalid